



SYSTEM AND METHOD FOR VERIFYING HARDWARE DESCRIPTION

Background of the Invention

1. Field of the Invention

5 The present invention relates to a—
~~systems~~systems and a ~~method~~methods for verifying
hardware description. More particularly, the present
invention relates to a system and a method for
verifying a hardware description, for verifying
10 discrepancies between a result~~program~~ obtained by
compiling hardware description in a programming
language and behavioral synthesis of the hardware
description.

2. Description of the Related Art

15 Conventionally, a hardware description
language (HDL) is used for the design of hardware, and
a software program for controlling the hardware is
written in a programming language. The ~~total~~—
verification of the hardware and the software program
20 as a whole is generally carried out using a simulator
called a co-simulator, which is operable with both ~~of~~—
a hardware description language and a programming
language. At this time, a software model ~~is~~—
~~simulated~~simulates ~~on~~ a hardware model to be verified.
25 Thus, the hardware model is verified. However, when
the hardware description language for the hardware
model and the programming language for the software

model are cooperated with each other for verifying the hardware model, the verification time is significantly dependent on the speed of the hardware simulation. For this reason, for high speed verification, both of
5 the hardware model and the software model are described in programming languages without using the hardware description language for high speed verification. For example, the hardware model is described in an advanced programming language using
10 standard software development tools and advanced programming languages, such as C, C++, or Java, ~~which is commonly used for developing types of software.~~

Generally, two major steps shown in Fig. 1 are necessary for the design of a new circuit. ~~Such~~
15 These steps are a function verifying step S101 and a behavioral synthesizing step S102. At the function verifying step S101, a source program in a programming language for the hardware descriptions is compiled into an executable program, and functions are verified
20 using the executable program. At ~~the~~ step S102, the hardware descriptions in the hardware language are synthesized in operation into a register transfer level (RTL) description.

However, in this case, another problem has
25 arisen. The programming language ~~for software development is~~ does not always correspond applicable to the hardware description design. For example, the

simultaneous parallel operations of circuits cannot be described in a general programming language. For the purpose ~~that of~~ the programming language ~~for software development meets~~ meeting the requirements for the

5 hardware design, the programming language is modified so as to extend operation specifications. However, there is a possibility ~~that any of a~~ discrepancy ~~is in~~ logic interpretation between the hardware ~~descriptions as represented in the program programming language~~

10 with an extended function and the hardware descriptions in the hardware language. Accordingly, it is necessary to verify the ~~equality~~ equivalence between the hardware ~~descriptions as represented in~~ the extended ~~program programming language~~ and the

15 hardware descriptions in the hardware description language. In this case, even if the discrepancy is founded, the reason why the discrepancy is caused and the ~~sentence~~ statement from which the discrepancy is caused are not manifest. ~~More~~ Additional time is

20 required to ~~make~~ determine the reason ~~and the description manifest.~~

In conjunction with the above description, an apparatus for designing an electronic circuit using a ~~program language~~ programming language is disclosed in

25 Japanese Laid Open Patent Application (JP-A-Heisei 10-149382). In this reference, a specific process section (2) sequentially specifies first portions to

be controlled, of a program describing a circuit behavior in a general ~~program-programming~~ language. A converting process section (3) converts the first portions into a program (4) using a general ~~program-~~ 5 ~~languageprogramming language~~ so as to operate as a state machine. Subsequently, a program producing process section (5) extracts second portions which circuits in the general ~~program-languageprogramming language~~ are operated in parallel and produces a 10 program (6) to access all the second portions.

Further, an extracting process section extracts a first accessing portion to hardware from a control program for controlling circuits described in a conversion and production program which is composed of 15 the converted program (4) and the produced program (6).

A first adding process section inserts immediately before the first accessing portion, a program for detecting an executing time of the first accessing portion and a program for measuring an execution time 20 period between the first accessing portion and a second accessing portion immediately before the first accessing portion. A second adding process section inserts immediately before the first accessing portion, a program operating the conversion and production 25 program for clocks corresponding to the execution time period obtained by the program inserted by the first adding process section.

Summary of the Invention

Therefore, an object of the present invention is to provide a system and a method for verifying hardware description while a portion of a source program for hardware description, ~~where it~~which portions interpretation is different between the programming language and a hardware description language is detected.

Another object of the present invention is to provide a system and a method for verifying the hardware description, where the verification of ~~equality~~equivalence can be omitted which is necessary when logic interpretation is not ~~equal~~equivalent between the programming language and the hardware description language.

In an aspect of the present invention, a hardware description verifying system includes a storage unit, an output unit and a processor. The storage unit stores a source program for hardware description in a ~~program~~programming language. The processor detect a portion of the source program different in logic interpretation between a ~~case of~~compiling the compiled source program using a compiler and a ~~case of~~behavioral synthesis produced unit, and outputs existence of the source program portion to the output unit.

The processor may detect the source program portion in which a variable of a register type is referred to at a clock timing after substitution to the variable at the clock timing. In this case, the
5 processor may include a signal list storage section and a processing section. The processing section sequentially reads out ~~sentences~~statements of the source program, and deletes a storage content in the signal list storage section when the read out
10 ~~sentence~~statement indicates a clock boundary. Also, the processing unit stores the variable in the signal list storage section when the read out ~~sentence~~statement indicates the substitution to the variable at the clock timing, and outputs the
15 existence of the read out ~~sentence~~statement to the output unit when the read out ~~sentence~~statement refers to the variable which is stored in the signal list storage section.

Also, the processor may detect the source
20 program portion in which substitution to a variable of a non-overwrite type is carried out twice or more at a clock timing. In this case, the processor may include a signal list storage section and a processing section. The processing section sequentially reads out
25 ~~sentences~~statements of the source program, and deletes a storage content in the signal list storage section when the read out ~~sentence~~statement indicates a clock

boundary. Also, the processing unit stores the variable in the signal list storage section when the read out ~~sentence~~statement indicates the substitution to the variable at the clock timing and the variable
5 is not stored in the signal list storage section, and outputs the existence of the read out ~~sentence~~statement to the output unit when the read out ~~sentence~~statement indicates the substitution to the variable at the clock timing and the variable is
10 stored in the signal list storage section.

Also, the processor may detect the source program portion in which a variable for a non-register type is referred to at a clock timing without substitution of the value to the variable at the clock
15 timing. In this case, the processor may include a signal list storage section and a processing section. The processing unit sequentially reads out ~~sentences~~statements of the source program, and deletes a storage content in the signal list storage section
20 when the read out ~~sentence~~statement indicates a clock boundary. Also, the processing unit stores the variable in the signal list storage section when the read out sentence indicates the substitution to the variable at the clock timing, and outputs the
25 existence of the read out ~~sentence~~statement to the output unit when the read out ~~sentence~~statement refers to the variable which is not stored in the signal list

storage section, at the clock timing.

Also, the processor may detect the source program portion in which a variable of a wiring line is referred to at a clock timing and then a value is substituted to the variable at the clock timing. In this case, the processor may include a signal list storage section and a processing section. The processing unit sequentially reads out ~~sentences~~statements of the source program, and deletes a storage content in the signal list storage section when the read out ~~sentence~~statement indicates a clock boundary. Also, the processing unit stores the variable in the signal list storage section when the read out sentence indicates the substitution to the variable at the clock timing, and outputs the existence of the read out ~~sentence~~statement to the output unit when the read out ~~sentence~~statement refers to the variable which is not stored in the signal list storage section, at the clock timing.

Also, the processor detects the source program portion in which a logical operator is used and a right operand of the operator includes a variable with substitution.

In another aspect of the present invention, a hardware description verifying method is attained by (a) detecting a portion of a source program different in logic interpretation between a case of compiling

the source program using a compiler and a case of behavioral synthesis, the source program for hardware description being described in a ~~program~~ languageprogramming language; and by (b) alarming
5 existence of the source program portion.

The source program portion may be a portion in which a variable of a register type is referred to at a clock timing after substitution to the variable at the clock timing. In this case, the (a) detecting
10 may be attained by sequentially reading out ~~sentences~~statements of the source program; by deleting a storage content in a signal list storage section when the read out ~~sentence~~statement indicates a clock boundary; by storing the variable in the signal list
15 storage section when the read out ~~sentence~~statement indicates the substitution to the variable at the clock timing; and by detecting the read out ~~sentence~~statement as the source program portion when the read out ~~sentence~~statement refers to the variable
20 which is stored in the signal list storage section.

Also, the source program portion may be a portion in which substitution to a variable of a non-overwrite type is carried out twice or more at a clock timing. In this case, the (a) detecting may be
25 attained by sequentially reading out ~~sentences~~statements of the source program; by deleting a storage content in a signal list storage section

when the read out ~~sentence~~statement indicates a clock boundary; by storing the variable in the signal list storage section when the read out ~~sentence~~statement indicates the substitution to the variable at the
5 clock timing and the variable is not stored in the signal list storage section; and by detecting the source program portion when the read out ~~sentence~~statement indicates the substitution to the variable at the clock timing and the variable is
10 stored in the signal list storage section.

Also, the source program portion may be a portion in which a variable for a non-register type is referred to at a clock timing without substitution of the value to the variable at the clock timing. In
15 this case, the (a) detecting may be attained by: sequentially reading out ~~sentences~~statements of the source program; by deleting a storage content in a signal list storage section when the read out ~~sentence~~statement indicates a clock boundary; by
20 storing the variable in the signal list storage section when the read out ~~sentence~~statement indicates the substitution to the variable at the clock timing; and by detecting the source program portion when the read out sentence refers to the variable which is not
25 stored in the signal list storage section, at the clock timing.

Also, the source program portion may be a

portion in which a variable of a wiring line is referred to at a clock timing and then a value is substituted to the variable at the clock timing. In this case, the (a) detecting may be attained by

5 sequentially reading out ~~sentences~~statements of the source program; by deleting a storage content in the signal list storage section when the read out ~~sentence~~statement indicates a clock boundary; by

10 storing the variable in the signal list storage section when the read out ~~sentence~~statement indicates the substitution to the variable at the clock timing; and by detecting the source program portion when the read out ~~sentence~~statement refers to the variable which is not stored in the signal list storage section,

15 at the clock timing.

Also, the processor detects the source program portion in which a logical operator is used and a right operand of the operator includes a variable with substitution.

20 In still another aspect of the present invention, a program is provided for a hardware description verifying method which may be attained by (a) detecting a portion of a source program different in logic interpretation between a ~~case of~~

25 ~~compiling~~compiled the source program using a compiler and a ~~case of~~ behavioral synthesis produced unit, the source program for hardware description being

described in a ~~program-programming~~ language; and by
(b) alarming existence of the source program portion.

Brief Description of the Drawings

5 Fig. 1 is a diagram showing a problem in a
conventional technique;

Fig. 2 is a block diagram showing the
structure of a hardware description verifying system
according to an embodiment of the present invention;

10 Figs. 3 and 4 are circuit diagrams showing an
example of register type variables;

Fig. 5 is a circuit diagram showing an
example of a non-overwrite type variable;

15 Fig. 6 shows a procedure of detecting the
presence of the register type verification object;

Fig. 7 is a diagram showing a verifying
result of a source program for the register type
verification object;

20 Fig. 8 shows a circuit diagram showing an
example of a register type verification object;

Fig. 9 shows a procedure of detecting the
presence of the non-overwrite type verification
object;

25 Fig. 10 is a diagram showing a verifying
result of a source program for the non-overwrite type
verification object;

Fig. 11 shows a circuit diagram showing an

example of a non-overwrite type verification object;

Fig. 12 is a diagram showing a verifying result of a source program for the non-register type verification object;

5 Fig. 13 is a diagram showing a cause of the verifying result;

Fig. 14 shows a procedure of detecting the presence of the non-register type verification object;

Fig. 15 is a diagram showing a verifying
10 result of a source program for the wiring line type verification object;

Fig. 16 shows a circuit diagram showing an example of a wiring line type verification object;

Fig. 17 shows a procedure of detecting the
15 presence of the wiring line type verification object;

Fig. 18 is a diagram showing a verifying result of a source program for an operator type verification object; and

Fig. 19 shows a procedure of detecting the
20 presence of the operator type verification object.

Description of the Preferred Embodiments

Hereinafter, a hardware description ~~verifying~~
verification system of the present invention will be
25 described below in detail with reference to the attached drawings.

Fig. 2 is a block diagram showing the

structure of the hardware description verifying system according to an embodiment of the present invention. Referring to Fig. 2, the hardware description verifying system in the embodiment is composed of an input unit 3, a storage unit 4, a hardware description verifying section 1, an output unit 7, and a drive unit 9. The hardware description verifying section 1 includes a signal list storage section 2.

A source program written in a ~~program~~
10 programming language for a circuit and compiled using a compiler is inputted from the input unit 3 to the storage unit 4. Also, a control program executed by the hardware description verifying section 1 is read out from a recording medium by the drive unit 9 and
15 loaded on the verifying section 1.

The hardware description verifying section 1 executes the control program to ~~receive~~obtain a source program for the hardware description of the circuit from the storage unit 4 and to execute the
20 verifying operation of the source program 5. When an error is found in a specific portion of the source program, the hardware description verifying section 1 outputs to the output unit 7 a warning signal 6 indicative of the presence of the specific portion
25 together with the location of the specific portion and a corresponding portion of the hardware description model obtained by behavioral synthesis of the hardware

descriptions, if the hardware description model is stored in the storage unit 4. The output unit 7 displays the alarm signal 6, the location of the specific portion and the corresponding portion of the hardware description model. The signal list storing section 2 stores and outputs a signal 8 indicative of a verification object variable.

The object to be verified is classified into five types in the present invention. The five types are referred to as a register type, a non-overwrite type, a non-register type, a wiring line type, and an operator depending type.

```
clock ()  
15 x=a+b  
clock ()  
y=c-d
```

The above source program includes variables x and y. The variable x is described by referring to the values of a and b in a hardware description model. The variable y is also described by referring to the values of c and d in the hardware description model. In Fig. 3, the values of a and b are supplied to an adder and the addition result (a+b) of the adder is latched as the variable x by a register 13 in synchronous with a clock signal 11. Also, in Fig. 4, the values of c and d are supplied to a subtractor and

the subtraction result (c-d) of the subtractor is latched as the variable y by a register 14 in synchronous with a clock signal 12. Thus, the variables x and y are updated in synchronous with
5 clock signals 11 and 12. The two variables x and y are not updated at the timing when the values of a and b or c and d are supplied. The variable x and y are updated in synchronization with a clock description after the allocation of the data. The variable
10 updated in synchronization with the clock description is referred to as a variable of the register type in the present invention. In this way, latches and registers in a circuit can be expressed with those variables.

15

1. Example of register type verification object:

clock ();

x=a+b;

y=x+t;

20

clock ();

In the above program, the ~~sentences~~
statements x = a+b and y = x+t are provided between
clock description ~~sentences~~statements, clock (),
corresponding to clock signals. The addition result
25 of the values of a and b is substituted to the
variable x in synchronization with the clock signal
and then the variable x is substituted in the other

variable y. When the variable is described or defined twice at the single clock signal, it is interpreted that the variable x does not have the same data in the behavioral synthesis of hardware descriptions in the hardware description language. However, a compiler interprets that the variable x has the same data. When the variable is updated in synchronization with the clock signal and then substituted to another variable in a sentencestatement, the sentencestatement is referred to as a register type verification object.

Fig. 7 shows another example of the register type verification object. The hardware description of the source program containing a sentencestatement for the register type verification object is:

```
15      int t;

      reg x, y;

      x=0;          first text
      clock();      second text
      x=1;          third text
20      t=3;          fourth text
      y=x+t;        fifth text
      clock();      sixth text
```

where the sentencestatement "reg x, y" and the second and sixth sentencestatements are written in the programming language with the extended function.

Fig. 6 shows a procedure of detecting the presence of the register type verification object.

Referring to Fig. 6, the hardware description verifying section 1 sequentially reads out the sentence statements of the source program from the storage unit 4 one by one (step S102), ~~when the~~
5 ~~sentences are remained~~ so long as statements remaining (step S101). The first sentence statement is "x=0". When the first sentence statement indicates a clock boundary, all the variables stored as data signals ~~stored in on~~ the signal list storing section 2 are
10 deleted. Since the first sentence statement is not the clock boundary, the procedure goes from the step S102 to a step S105. It is checked at the step S105 whether the read out sentence statement refers to any variable stored in the signal list storing section 2.
15 When the read out sentence statement refers to the variable stored in the signal list storing section 2, there is a possibility that an unintentional behavior is carried out in the software execution. Therefore, the hardware description verifying section 1 outputs
20 an alarm signal 6 to the output unit 7. Then the procedure advances to a step S107. No variable is now stored in the signal list storing section 2 and thus the procedure advances directly to the step S107. It is then checked at step S107 whether the substitution
25 to the register type variable (x) is present. Since the first sentence statement includes the substitution to the register type variable, the procedure advances

to a step S108.

At the step S108, when the substitution to the register type variable is present, the register type variable in the senteneestatement is stored in
5 the signal list storing section 2. When the register type variable x in the first senteneestatement has been stored in the signal list storing section 2, the procedure returns back to the step S101.

Next, the second senteneestatement "clock ()"
10 is inputted to the hardware description verifying section 1. Since the second senteneestatement indicates a clock boundary, the variables stored in the signal list storing section 2 are deleted at the step S104. Then, the procedure then returns back to
15 the step S101.

The third senteneestatement "x=1" is inputted to the hardware description verifying section 1. Since the third senteneestatement indicates no clock boundary, the procedure jumps to the step S105. Since
20 no variable is registered in the signal list storage section 2, the procedure goes via the step S105 to the step S107 where it is determined that there is substitution to the register type variable x. Accordingly, the step S108 is executed to register the
25 variable x in the signal list storage section 2.

Next, the fourth senteneestatement "t=3" is inputted to the hardware description verifying section

1. Since the fourth ~~sentence~~statement indicates no clock boundary, the procedure jumps to the step S105. Since the fourth ~~sentence~~statement does not refer to the variable x, the procedure goes from the step S105
5 to step S107 where it is determined that the substitution to the register type variable is not present. Accordingly, the value of t is not , registered in the signal list storage section 2 and the procedure returns back to the step S101.

10 The fifth ~~sentence~~statement "y=x+t" is inputted to the hardware description verifying section 1. Since the fifth ~~sentence~~statement indicates no clock boundary, the procedure jumps to the step S105. Since the variable y in the fifth ~~sentence~~statement
15 refers to the variable x registered in the signal list storing section 2, an alarm is outputted at the step S106. Then, it is determined at the step S107 that the substitution to the register type variable y is present so that the variable y in the fifth
20 ~~sentence~~statement is registered in the signal list storing section 2 as a signal y at the step S108. Next, the sixth ~~sentence~~statement is inputted to the hardware description verifying section 1 and the two variables x and y are deleted from the signal list
25 storing section 2.

 In the third and fifth ~~sentence~~statements between the clock boundary descriptions, the variable

y in the fifth ~~sentence~~statement refers to the third ~~sentence~~statement which is another ~~sentence~~statement. In this case, interpretation by the compiler for the C language is different from interpretation by the hardware description language (HDL), as shown in Fig. 7. Even when the source program is correctly written in a computer language, the interpretation may be different between the extended C language and the HDL language. As an example of such a case, there is a case that the variable substituted at a clock timing is referred to in another ~~sentence~~statement at the same clock timing.

The value of 0 is substituted to the variable x in the HDL language at the clock timing of the second ~~sentence~~statement. On the other hand, the value of 1 is substituted to the variable x in the C language. Therefore, y=3 in the HDL and y=4 in the C language. The ~~sentence~~statements before and after the clock boundary of which a physical operation is executed by use of a register at the timing of a clock signal are interpreted and calculated differently between the C language and the HDL language. Since the interpretation is different, the fifth ~~sentence~~statement produces different results by referring to the other ~~sentence~~statement.

2. Non-overwrite type verification object:

```
clock ();  
z=a+b;  
z=c+d;  
clock ();
```

5 According to the above program, two ~~sentences~~statements for the variable z are described by referring to the values of a and b and c and d in the hardware, respectively. It is defined that ~~z~~z is a variable which is not overwritten at the same clock
10 timing in the extended C language. Those variables are used for expressing a multiplexer in the actual circuit. One of the two variables z is described with the values of a and b while the other with the values of c and d. In the compiler, an upper one of the
15 ~~sentences~~statements for the two variables z is substituted in the lower equation. Hence, the variable z of the lower ~~sentence~~statement is valid. As shown in Fig. 5, the HDL interprets in behavioral synthesis that one of (a+b) and (c+d) is selected by a
20 multiplexer (MX) 15. With the non-overwrite type, the ~~twice~~two substitutions to the variable at the same clock timing are not permitted.

Fig. 9 illustrates a procedure of detecting the presence of a non-overwrite type verification
25 object. Fig. 10 shows an example of the non-overwrite type verification object. The hardware description of the non-overwrite type verification object in the

extended C language of the program is:

```
        ter z, t;
        z=0;          first sentenestatement
        clock ();     second sentenestatement
5        z=1;          third sentenestatement
        t=3;          fourth sentenestatement
        z=t+2;        fifth sentenestatement
        clock ();     sixth sentenestatement
```

where "ter z, t" and the second and sixth
10 ~~statementssentences~~ are described in the extended C
programming language.

The hardware description verifying section 1
inputs each ~~sentene~~statement of the hardware
description from the storage unit 4 (step S202), when
15 ~~the sentences are~~there are statements remaining.
~~remained~~ (step S201). The first ~~sentene~~statement is
"z=0". When the first ~~sentene~~statement indicates a
clock boundary (Yes at a step S203), all the signals
stored in the signal list storing section 2 are
20 deleted (step S204). As the first ~~sentene~~statement
is not the clock boundary, the procedure goes from the
step S202 to a step S205.

It is checked at the step S205 whether the
inputted ~~sentene~~statement includes substitution to
25 the variable stored in the signal list storing section
2. By now, no variable is registered and the
procedure jumps to a step S207. When the substitution

is present, an alarm is outputted before the procedure moves to the step S207.

It is then checked at the step S207 whether the substitution to the non-overwrite type signal z is present. Since the first ~~sentence~~statement includes the substitution to the non-overwrite type signal z, the procedure advances to a step S208. At the step S208, when the substitution to the non-overwrite type signal is present, the non-overwrite type variable substituted in the ~~sentence~~statement is stored in the signal list storing section 2. Therefore, when the non-overwrite type variable z in the first ~~sentence~~statement has been stored in the signal list storing section 2, the procedure returns back to the step S201.

Next, the second ~~sentence~~statement, clock (), is inputted to the hardware description verifying section 1. Since the second ~~sentence~~statement indicates a clock boundary, the variables stored in the signal list storing section 2 are all deleted. In this example, the variable z is deleted at the step S204. Then, the procedure then returns back to the step S201.

The third ~~sentence~~statement "z=1" is inputted to the hardware description verifying section 1. As the third ~~sentence~~statement indicates no clock boundary, the procedure jumps to the step S205. By

now, no variable is registered and the procedure goes from the step S205 to the step S208 via the step S207 where the signal z is registered in the signal list storage section 2.

5 Next, the fourth ~~sentence~~statement "t=3" is inputted to the hardware description verifying section 1. Since the fourth ~~sentence~~statement indicates no clock boundary, the procedure jumps to the step S205. Since the signal list storage section 2 holds z but
10 not t, the procedure goes from the step S205 to the step S207. Since the substitution to the non-overwrite type signal is present, the variable t is registered. Then, the procedure returns back to the step S201.

15 Next, the fifth ~~sentence~~statement "z=t+2" is inputted to the hardware description verifying section 1. Since the fifth ~~sentence~~statement indicate no clock boundary, the procedure jumps to the step S205. Since the variable z in the fifth ~~sentence~~statement is
20 the variable registered in the signal list storing section 2, an alarm is outputted at a step S206. Then, it is determined at the step S207 that the substitution to the non-overwrite type variable z is present, and then, the variable z in the fifth
25 ~~sentence~~statement is registered in the signal list storing section 2 at the step S208. Next, the sixth ~~sentence~~statement is inputted to the hardware

description verifying section 1 and the two variables z are deleted from the signal list storing section 2.

In the third and fifth ~~sentene~~statements between the clock boundary descriptions, the variable
5 z in the fifth ~~sentene~~statement may be overwritten by the variable z in the third ~~sentene~~statement. In such a case, interpretation in the compiler for the extended C language is different from that in the HDL, as shown in Fig. 10. As shown in Fig. 11, it is
10 unknown in the HDL whether either of the third or fifth ~~sentene~~statement is valid, so that the variable z at the clock timing is not defined, while the values of 1 and 5 are substituted for the third
~~sentene~~statement and the fifth ~~sentene~~statement in
15 the C language, respectively. The sentences before and after the clock boundary of which the physical operation is executed at the timing of a clock signal are interpreted differently between the C language and the HDL language or may be not calculated. Depending
20 on the interpretation as either overwriting or unknown, the result of the operation will be different.

As described above, the values remains unchanged after the substitution to the variable and are updated at once upon the description of clock
25 boundary in the register type. On the other hand, one variable cannot be substituted two or more times between any two clock boundary descriptions in the

non-overwrite type. Since the extended C programming language in which a clock boundary ~~statement~~statement can be introduced includes elaborate physical descriptions, there are cases that intentions of the designer fail to be expressed in the physical description. In the present invention, a portion of the hardware description where the ~~equality~~equivalent may be lost is detected, without examining the ~~equality~~equivalent between the functional verification of the hardware descriptions and the functional verification of the behavior synthesis of hardware descriptions. Thus, the designer or user can rewrite the detected portion in the source program to avoid different interpretations. The rewritten program contains no discrepancies in the interpretation and the verification of the equality can be omitted.

The other types described below are also defined equally and if a discrepancy is found, an alarm is outputted. Fig. 12 illustrates an example of the non-register type verification object. The hardware description of the non-register type verification object in an extended C language of the program is as follows:

25

3. Non-register type verification objects:

t=3;

```
clock ();
```

```
z=t+2;
```

```
clock ();
```

In the program, the value of t is described
5 outside of the clock period between two clock
boundaries. The value of t for which 3 is substituted
is valid only inside the clock period where the value
of t is defined. The value of 3 for the variable t is
invalid when the variable t is referred to over the
10 clock boundary. Such a variable t is called the non-
register type.

The value substituted previously can be used
in the C language. However, it is interpreted in the
hardware description language (HDL) that the
15 substitution to the variable t is not present in the
same step. Therefore, the value to be referred to
depends on a synthesizing tool, as shown in Fig. 13.
The value of the variable z which refers to the non-
register type variable t whose value is not
20 substituted before the referencing operation in the
same step may be different between the extended C
language and the hardware description language. As
shown in Fig. 12, z=5 is in the extended C language
but z is unknown in the hardware description language.
25 Since the third ~~sentence~~statement "z=t+2" refers to
the non-register type variable t not registered in the
signal list storage section 2, an alarm is outputted

through steps S305 and S306 shown in Fig. 14. When it is determined at a step S307 that the substitution to the non-register type signal is present, the non-register type signal t is registered in the signal
5 list storage section 2 at a step S308. The other steps are same as those shown in Fig. 6.

4. Wiring line type verification object

The hardware description of the wiring line type verification object in an extended C language of
10 the program is as follows:

```
        t=3;  
        clock ();  
        z=t+2;  
        t=1;  
15      clock ();
```

In the above source program, the variable t is described two times in the same clock period between two clock boundaries. The variable t whose substituted value is reflected in all the references
20 at the clock period is called of the wiring line type. The signal represents a pattern of signal wiring line in an actual circuit.

The value substituted previously can be used in the C language. On the other hand, only the value
25 substituted to variable t in the same clock period can be used in the hardware description language. As shown in Fig. 16, the substituted value of the

variable t is different between the extended C language and the hardware description language. The value is either 3 equal to the previous value in the extended C language or 1 equal to the current value in the hardware description language. Referring to Fig. 15, z=5 and t=1 are defined in the C language while z=3 and t=1 in the hardware description language. Since the third ~~sentence~~statement "z=t+2" refers to the wiring line type variable t not registered in the signal list storage section 2, an alarm is outputted through steps S405 and S406 shown in Fig. 17. When it is determined at a step S407 that the substitution to the wiring line type signal is present, the wiring line type signal t is registered in the signal list storage section 2 at a step S408. The other steps are same as those shown in fig. 6. The verifying result is shown in Fig. 15.

5. Operator type verification object

Fig. 18 illustrates an example of the operator type verification object. The hardware description of the operator type verification object in an extended C language of an original program is as follows:

```
25      i=0;
        a=0;
        clock ();
```

```
    if (i>0&&a++){  
        i=0;  
    }  
    clock ();
```

5 where a++ means that a is incremented when being
evaluated. With the operator && in the program is in
the C language, the right operand a++ is evaluated
when the left operand (i>0) is true. In the hardware
description language, both of the left and right
10 operands of the operator && are evaluated in parallel
regardless of whether the left operand is true or not.
When the left operand of the operator && is not true,
the evaluation of the right operand may be different
between the C language and the hardware description
15 language. As shown in Fig. 19, when the operator &&
is used at a step S503 and the description for
updating the variable is present in the right operand
of the operator && (step S504), an alarm is outputted
at a step S505. Also, when the operator && is
20 replaced by another operator for evaluating the right
operand only if the left operand is not true, an alarm
can be outputted as shown in Fig. 19. The other steps
are same as those shown in Fig. 6.

In the system and method for verifying the
25 hardware description according to the present
invention, a portion of the description where its
interpretation is different between the programming

language and the hardware description language is detected and an alarm is outputted to indicate that the portion may interrupt the operation of simulating the functional verification, hence allowing the
5 program to be modified with much ease or eliminating the interruption. In particular, the verification of equality will successfully be omitted.

Abstract of the Disclosure

A hardware description verifying system includes a storage unit, an output unit and a processor. The storage unit stores a source program
5 for hardware description in a ~~program-programming~~ language. The processor detect a portion of the source program different in logic interpretation between a ~~case of compiling the~~compiled source program using a compiler and a ~~case of behavioral synthesis~~
10 produced module, and outputs existence of the source program portion to the output unit.